

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни
“АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ”
для студентів напряму підготовки
121 «Інженерія програмного забезпечення»
(всіх форм навчання)

2020

Методичні вказівки до лабораторних робіт з дисципліни «Алгоритми та структури даних» для студентів напряму підготовки 121 «Інженерія програмного забезпечення» (всіх форм навчання) / Н. О. Миронова, Л.Ю. Дейнега. – Запоріжжя: ЗНТУ, 2020. – 30 с.

Автори: Н. О. Миронова, к.т.н., доцент
Л.Ю. Дейнега, ст. викладач

Рецензент: Т.В. Федорончак, к.т.н., доцент

Відповідальний
за випуск: Н. О. Миронова, к.т.н., доцент

Затверджено
на засіданні кафедри
програмних засобів

Протокол №8
від “16” червня 2020 р.

ЗМІСТ

Вступ.....	4
Лабораторна робота №1 Алгоритми пошуку та сортування.....	5
Лабораторна робота №2 Алгоритми роботи зі структурами даних	9
Лабораторна робота №3 Алгоритми теорії чисел	15
Лабораторна робота №4 Динамічне програмування.....	19
Лабораторна робота №5 Алгоритми обходу графу.....	22
Лабораторна робота №6 Алгоритми пошуку найкоротших шляхів у графах	25
Лабораторна робота №7 Дерево відрізків	28
ЛІТЕРАТУРА	30

ВСТУП

Метою лабораторних робіт є вивчення теорії алгоритмів та практичного засвоєння вміння використовувати дані алгоритми в процесі розв'язання практичних завдань шляхом розроблення програм з використанням мови програмування C++.

Кожному студенту при підготовці до виконання лабораторної роботи потрібно ознайомитись з методичними вказівками та конспектом лекцій по даному розділу. Виконання лабораторної роботи розбивається на два етапи. На першому етапі необхідно ознайомитись з конспектом лекцій. На другому етапі виконується індивідуальне завдання до лабораторної роботи. Наприкінці заняття результати роботи подаються викладачу для перевірки. Звіт з лабораторної роботи оформлюється на стандартних аркушах паперу формату А4 (297x210мм).

Лабораторна робота обов'язково захищається на наступному після виконання роботи навчальному занятті. За несвоєчасний захист роботи оцінка знижується. Захист лабораторних робіт проводиться під час навчальних занять. Студент не допускається до виконання наступної роботи, якщо має дві незахищені роботи.

ЛАБОРАТОРНА РОБОТА №1

АЛГОРИТМИ ПОШУКУ ТА СОРТУВАННЯ

1.1 Мета роботи

Вивчити алгоритми бінарного пошуку та сортування.

1.2 Теоретичні відомості

Алгоритми пошуку. Одна з тих дій, які найбільш часто зустрічаються в програмуванні – пошук. Існує декілька основних варіантів пошуку, і для них створено багато різноманітних алгоритмів.

Задача пошуку – відшукати елемент, ключ якого рівний заданому аргументу пошуку. Отриманий в результаті цього індекс забезпечує доступ до усіх полів виявленого елемента.

Послідовний (лінійний) пошук. Найпростішим методом пошуку елемента, який знаходиться в невідсортованому наборі даних, за значенням його ключа є послідовний перегляд кожного елемента набору, який продовжується доти, поки не буде знайдений потрібний елемент. Якщо переглянуто весь набір, і елемент не знайдений – тоді шуканий ключ відсутній в наборі. Цей метод ще називають методом повного перебору.

Для послідовного пошуку в середньому потрібно $N/2$ порівнянь, де N – кількість елементів у наборі даних. Таким чином, порядок алгоритму – лінійний – $O(N)$. Якщо елемент знайдено, то його індекс мінімальний з можливих, тобто це перший з таких елементів. Рівність $i=N$ засвідчує, що елемент відсутній.

Якщо немає додаткових вказівок про розташування необхідного елемента, то природним є послідовний перегляд масиву із збільшенням тієї його частини, де бажаного елемента не знайдено. Такий метод називається лінійним пошуком.

Бінарний пошук елемента в масиві.

Якщо є масив, що містить впорядковану послідовність даних, то дуже ефективним є бінарний пошук, або метод ділення навпіл. Відповідно до назви, його ідея полягає у послідовному поділенні масиву навпіл і визначенні, в якій із половин знаходиться шуканий елемент. Для цього, завдяки відсортованості даних у масиві, достатньо порівняти шуканий елемент із центральним.

Змінні L і R містять відповідно ліву і праву межі відрізка масиву, де міститься потрібний елемент. Починаємо завжди з дослідження середнього елемента відрізка. Позначимо m – середній елемент, який дорівнює $(L + R) / 2$. Якщо шукане значення менше від середнього елемента, ми переходимо до пошуку у лівій половині відрізка, де всі елементи менші від тільки що перевіреного. Іншими словами, значенням R стає $(m-1)$, і на наступній ітерації працюємо з половиною масиву. В протилежному випадку елемент потрібно шукати в правій половині відрізка, тому зсуваємо ліву границю пошуку в середину: L стає $m+1$. Отже, в результаті кожної перевірки ми вдвічі звужуємо область пошуку. Порядок алгоритму бінарного пошуку – $O(\log N)$.

Матеріали з цієї теми подані в [1, 2].

Пошук методом Фібоначчі. Він працює швидше, ніж бінарний пошук, оскільки замість операцій ділення, що застосовуються у попередньому методі, використовує операції додавання та віднімання. Суть методу полягає у визначенні наступного елемента для порівняння з числами Фібоначчі. Зменшення індекса означає перехід по попереднього числа, збільшення - перехід до наступного. Числа Фібоначчі формуються на основі додавання двох попередніх чисел, де перше і друге число дорівнюють 1. Тобто, можна скласти таку послідовність чисел: 1,1,2, 3, 5, 8,13,21,34...

1.3 Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити алгоритми бінарного пошуку та сортування.

2. Виконати індивідуальне завдання згідно з варіантом (табл. 1.1). Написати програму, що реалізує алгоритм бінарного пошуку, та програму з використанням алгоритму бінарного пошуку `binary_search()` бібліотеки STL. Порівняти результати програм. Оцінити обчислювальну складність та час виконання алгоритмів.

3. Реалізувати алгоритм сортування згідно з варіантом (табл. 1.2). Написати програму, що реалізує алгоритм сортування, та програму з використанням алгоритму сортування `sort()` бібліотеки STL з використанням функцій компараторів. Порівняти результати

програм. Оцінити обчислювальну складність та час виконання алгоритмів.

4. Оформити звіт. Звіт з лабораторної роботи повинен містити: мету роботи; завдання на виконання роботи; блок-схеми програм; тексти програм; результати роботи; висновки.

5. Відповісти на контрольні запитання.

1.4 Індивідуальні завдання

Таблиця 1.1 – Завдання на тему «Бінарний пошук»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/uk/problems/7329
2	https://www.e-olymp.com/uk/problems/3968
3	https://www.e-olymp.com/uk/problems/3966
4	https://www.e-olymp.com/uk/problems/3967
5	https://www.e-olymp.com/uk/problems/3969
6	https://www.e-olymp.com/uk/problems/3970
7	https://www.e-olymp.com/uk/problems/6356
8	https://www.e-olymp.com/uk/problems/4035
9	https://www.e-olymp.com/uk/problems/5149
10	https://www.e-olymp.com/uk/problems/6008

Таблиця 1.2 – Завдання на тему «Алгоритми сортування»

№	Алгоритми сортування
1	метод сортування підрахунком
2	метод сортування бульбашкою
3	метод сортування вставками
4	метод сортування злиттям
5	метод сортування вибором
6	метод швидкого сортування

1.5 Контрольні запитання

1. У чому полягає ідея лінійного пошуку? Сформулюйте умову припинення перегляду елементів.

2. У якому випадку може використовуватися бінарний пошук? У чому його основна ідея?

3. Сформулюйте алгоритм бінарного пошуку.

4. В чому полягає метод сортування підрахунком?

5. В чому полягає метод сортування бульбашкою?

6. В чому полягає метод сортування вставками?

7. В чому полягає метод сортування злиттям?

8. В чому полягає метод сортування вибором?

9. В чому полягає метод швидкого сортування?

ЛАБОРАТОРНА РОБОТА №2

АЛГОРИТМИ РОБОТИ ЗІ СТРУКТУРАМИ ДАНИХ

2.1 Мета роботи

Вивчити алгоритми створення та керування структурами даних однозв'язний та двозв'язний список, стеки та черги, множина, рядок.

2.2 Теоретичні відомості

Динамічні структури даних в процесі існування в пам'яті можуть змінювати не тільки число складових їх елементів, а й характер зв'язків між елементами. Динамічна структура даних характеризується тим, що вона не має імені, їй виділяється пам'ять в процесі виконання програми, кількість елементів структури може не фіксуватися, розмірність структури може змінюватися в процесі виконання програми, в процесі виконання програми може змінюватися характер взаємозв'язку між елементами структури. Для встановлення зв'язку між елементами динамічної структури використовуються показники, через які встановлюються явні зв'язки між елементами. Таке представлення даних в пам'яті називається пов'язаним. Порядок роботи з динамічними структурами даних наступний: 1) створити (відвести місце в динамічній пам'яті); 2) працювати за допомогою показника; 3) видалити (звільнити зайняте структурою місце).

До динамічних інформаційних структур відносять:

- однозв'язні списки;
- двозв'язні списки;
- стек;
- чергу;
- множину.

Елемент динамічної структури складається з двох полів: інформаційного поля (поля даних), в якому містяться дані; адресного поля (поля зв'язок), в якому містяться один або кілька показників, що зв'яже даний елемент з іншими елементами структури.

Списком називається впорядкована множина, що складається зі змінного числа елементів, до яких застосовані операції включення, виключення. Довжина списку дорівнює числу елементів, що містяться

у списку, список нульової довжини називається порожнім списком. Для зв'язування елементів у списку використовують систему покажчиків. У мінімальному випадку будь-який елемент лінійного списку має один покажчик, який вказує на наступний елемент у списку або є порожнім покажчиком, що інтерпретується як кінець списку. Структура, елементами якої служать записи з одним і тим самим форматом, пов'язані один з одним за допомогою покажчиків, що зберігаються у самих елементах, називають пов'язаним списком. Кожен список має особливий елемент, називаний покажчиком початку списку (головою списку), який зазвичай за змістом відмінний від інших елементів. В полі покажчика останнього елемента списку знаходиться спеціальна позначка NULL, яка свідчить про кінець списку.

З усього різноманіття пов'язаних списків можна виділити однозв'язані списки та двозв'язані списки. В основному вони відрізняються видом взаємозв'язку елементів і / або допустимими операціями.

Найбільш простою динамічною структурою є однозв'язний список, елементами якого служать об'єкти структурного типу. Однозв'язний список – це послідовність елементів, в кожному з яких зберігається значення і покажчик на наступний елемент списку. В останньому елементі покажчик на наступний елемент дорівнює NULL. Опис найпростішого елемента такого списку є наступним:

`struct ім'я_типу {інформаційне поле; адресне поле; };` де інформаційне поле – це поле будь-якого, раніше оголошеного або стандартного типу; адресне поле – це покажчик на об'єкт того ж типу, що і визначається структура, в нього записується адреса наступного елемента списку.

Двозв'язані списки – це структура даних, що складається з послідовності елементів, кожен з яких містить інформаційну частину і два покажчика на сусідні елементи. При цьому два сусідні елементи повинні містити взаємні посилання один на одного. Опис найпростішого елемента такого списку є наступним:

`struct ім'я_типу { інформаційне поле; адресне поле 1; адресне поле 2; };` де інформаційне поле – це поле будь-якого, раніше оголошеного або стандартного, типу; адресне поле 1 – це покажчик на об'єкт того самого типу, що і визначена структура, в нього записується адреса наступного елемента списку; адресне поле 2 - це покажчик на

об'єкт того самого типу, що і визначена структура, в нього записується адреса попереднього елемента списку. Основні операції, що виконуються над двозв'язним списком, ті самі, що й для однозв'язного списку. Тому що двозв'язний список більш гнучкий, ніж однозв'язний, то при включенні елемента у список, потрібно використовувати покажчик як на елемент, за яким відбувається включення, так і покажчик на елемент, перед яким відбувається включення. При виключення елемента зі списку потрібно використовувати як покажчик на сам виключається елемент, так і покажчики на попередній або наступний за виключається елементи. Але тому що елемент двоів'язного списку має два покажчика, то при виконанні операцій включення / виключення елемента треба змінювати 15 більше зв'язків, ніж в однозв'язному списку.

У списках доступ до елементів відбувається за допомогою адресації, при цьому доступ до окремих елементів не обмежений. Але існують також і такі спискові структури даних, в яких є обмеження доступу до елементів. Одним з представників таких спискових структур є стековий список або просто стек. Стек (англ. Stack – стопка) – це структура даних, в якій новий елемент завжди записується в її початок (вершину) і черговий елемент читається також завжди вибирається з її початку. У стеках використовується метод доступу до елементів LIFO (Last Input – First Output, "останнім прийшов – першим вийшов"). Найчастіше принцип роботи стека порівнюють зі стопкою тарілок: щоб взяти другу зверху, потрібно спочатку взяти верхню. Стек – це список, у якого доступний один елемент (одна позиція). Цей елемент називається вершиною стека. Взяти елемент можна тільки з вершини стека, додати елемент можна тільки у вершину стека.

Опис стека є наступним: `struct ім'я_типу { інформаційне поле; адресне поле; };` де інформаційне поле – це поле будь-якого раніше оголошеного або стандартного типу; адресне поле – це покажчик на об'єкт того ж типу, що і структура, в нього записується адреса наступного елемента стека.

Стек як динамічну структуру даних легко створити на основі лінійного списку. Оскільки робота завжди йде з заголовком стека, тобто не потрібно здійснювати перегляд елементів, видалення і вставку елементів у середину або кінець списку, то досить використовувати економічний по пам'яті лінійний однозв'язний

список. Для такого списку досить зберігати покажчик вершини стека, який вказує на перший елемент списку. Якщо стек порожній, то списку не існує, і покажчик набуває значення NULL.

Черга – це структура даних, яка є послідовністю елементів, що створена в порядку їх надходження. Кожен новий елемент розміщується в кінці черги; елемент, що стоїть на початку черги, вибирається з неї першим. У черзі використовується принцип доступу до елементів FIFO (First Input – First Output, "перший прийшов – першим вийшов"). У черзі доступні два елементи (дві позиції): початок черги і кінець черги. Помістити елемент можна тільки в кінець черги, а взяти елемент тільки з її початку. Прикладом може бути звичайна черга в магазині. Чергу як динамічну структуру даних легко створити на основі лінійного списку. Оскільки робота відбувається з обома кінцями черги, то переважно використовувати лінійний двоспрямований список. Хоча для роботи з таким списком досить мати один покажчик на будь-який елемент списку, тут доцільно зберігати два покажчики – один на початок списку (звідки вилучаємо елементи) і один на кінець списку (куди додаємо елементи). Якщо черга порожня, то списку не існує, і покажчики набувають значення NULL.

Множина - відсортований набір однотипних елементів, які не повторюються. Реалізувати таку структуру даних можна різними способами: за допомогою динамічного масива або лінійного списку.

2.3 Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити структури даних однозв'язний та двозв'язний список, дек, стек, черга, множина, рядок.
2. Виконати індивідуальне завдання згідно з варіантом.
3. Оформити звіт. Звіт з лабораторної роботи повинен містити: мету роботи; завдання на виконання роботи; текст програми; результати роботи; висновки.
4. Відповісти на контрольні запитання.

2.4 Індивідуальні завдання

1. Розробити програму, що реалізує структуру даних стек. Порівняти з реалізацією контейнера бібліотеки stack, розробивши

відповідну програму. Оцінити обчислювальну складність та час виконання алгоритмів.

2. Розробити програму, що реалізує структуру даних дек. Порівняти з реалізацією контейнера бібліотеки deque, розробивши відповідну програму. Оцінити обчислювальну складність та час виконання алгоритмів.

3. Розробити програму, що реалізує структуру даних черга. Порівняти з реалізацією контейнера бібліотеки queue, розробивши відповідну програму. Оцінити обчислювальну складність та час виконання алгоритмів.

4. Розробити програму, що реалізує структуру однозв'язний список. Порівняти з реалізацією контейнера бібліотеки list, розробивши відповідну програму. Оцінити обчислювальну складність та час виконання алгоритмів.

5. Розробити програму, що реалізує структуру двозв'язний список. Порівняти з реалізацією контейнера бібліотеки list, розробивши відповідну програму. Оцінити обчислювальну складність та час виконання алгоритмів.

6. Розробити програму, що реалізує структуру даних множина. Оцінити обчислювальну складність та час виконання алгоритмів. Порівняти з реалізацією контейнера бібліотеки set, розробивши відповідну програму.

7. Розробити програму, що реалізує структуру даних рядок символів, використовуючи однозв'язний список (елемент списку містить дані: один символ, та покажчик на наступний елемент списку). Порівняти з реалізацією контейнера бібліотеки string, розробивши відповідну програму. Оцінити обчислювальну складність та час виконання алгоритмів.

8. Розробити програму, що реалізує структуру даних рядок символів, використовуючи динамічний масив символів. Порівняти з реалізацією контейнера бібліотеки string, розробивши відповідну програму. Оцінити обчислювальну складність та час виконання алгоритмів.

2.5 Контрольні запитання

1. Як здійснюється доступ до динамічних структур з програмного коду?
2. Як зв'язуються між собою елементи динамічної структури?
3. Якого типу може бути поле даних в динамічній структурі?
4. За рахунок чого робота з динамічними даними уповільнює виконання програми?
5. У чому відмінність першого елемента однозв'язного (двозв'язного) списку від інших елементів цього ж списку?
6. У чому відмінність останнього елемента однозв'язного (двозв'язного) списку від інших елементів цього ж списку?
7. Чому при роботі з двозв'язним списком не обов'язкове позиціонування на перший елемент списку?
8. У чому принципові відмінності виконання основних операцій в односпрямованих і двоспрямованих списках?
9. Яке призначення деку, стеку, черги, списку?
10. Як створити дек, стек, чергу, список?
11. Назвіть базові операції над деком, стеком, чергою і списком.
12. З якою метою в програмах виконується перевірка на порожнечу стека і черги?
13. З якою метою в програмах виконується видалення стека і черги після закінчення роботи з ними?

ЛАБОРАТОРНА РОБОТА №3 АЛГОРИТМИ ТЕОРІЇ ЧИСЕЛ

3.1 Мета роботи

Вивчити алгоритми теорії чисел.

3.2 Теоретичні відомості

Теорія чисел розглядає властивості цілих чисел, тобто чисел класу $Z = 0, \pm 1, \pm 2, \dots$. Сума та добуток цілих чисел є цілими числами, але про результат ділення двох цілих чисел такого стверджувати вже не можна.

Для довільних цілих чисел a і b визначається, що b ділить a (a ділиться на b), якщо існує таке ціле число q , для якого виконується рівність: $a = b \cdot q$. Позначається цей факт так: $b \mid a$. При цьому b називається дільником a , число a за таких умов називається кратним b .

Будь-яке ціле d , яке одночасно ділить числа $a, b, \dots, l$, має назву спільного дільника цих чисел. Найбільший із всіх дільників має назву **найбільшого спільного дільника** (НСД) і позначається $d = (a, b, \dots, l)$. Якщо $(a, b, \dots, l) = 1$, то числа $a, b, \dots, l$ взаємно прості. Якщо кожне число із наведеного набору є взаємно простим із кожним іншим числом цього набору, то ці числа попарно прості. Попарно прості числа є одночасно і взаємно простими, але не навпаки.

Для чисел a і b процес послідовного ділення з остачею (a ділиться на b , потім b на r_1 , потім перша остачу на другу і т. д.) називається **алгоритмом Евкліда**:

$$a = bq_0 + r_1 \quad (0 < r_1 < b),$$

$$b = r_1q_1 + r_2 \quad (0 < r_2 < r_1),$$

$$r_1 = r_2q_2 + r_3 \quad (0 < r_3 < r_2),$$

$$\dots\dots\dots$$

$$r_{n-2} = r_{n-1}q_{n-1} + r_n \quad (0 < r_n < r_{n-1}),$$

$$r_{n-1} = r_nq_n \quad (0 < r_{n+1} < r_n)$$

Найменше спільне кратне (НСК) двох цілих чисел — найменше натуральне число, яке є кратним обох цих чисел.

Властивості НСК:

$\text{НСК}(a, b) = \text{НСК}(b, a)$ (перестановка аргументів не змінює НСК);

$\text{НСК}(a, b, c, d) = \text{НСК}(\text{НСК}(a, b), \text{НСК}(c, d))$;

$\text{НСК}(a, b) = |ab|/\text{НСД}(a, b)$, де $\text{НСД}(a, b)$ — найбільший спільний дільник чисел a, b .

Решето́ Ератосфе́на — простий стародавній алгоритм знаходження всіх простих чисел менших деякого цілого числа, що був створений давньогрецьким математиком Ератосфеном.

Якщо потрібно знайти всі прості числа менші за певне число N , виписуються всі числа від 1 до N .

Перше просте число - 2. Викреслюються всі числа більші 2, які діляться на 2 (4, 6, 8 ...).

Наступне число, яке залишилося незакресленим (3), є простим. Викреслюються всі числа більші 3 та кратні 3 (6, 9 ...).

Наступне незакреслене число (5) є простим. Викреслюються всі числа більші 5 та кратні 5 (10, 15, 20, 25 ...).

Повторюється операція поки не буде досягнуто число N :

Кожне наступне незакреслене число є простим. Викреслюються всі числа більші нього та кратні йому.

Числа, які залишилися незакресленими після цієї процедури - прості.

Бінарне піднесення до ступеня

Нехай $n = (\overline{m_k m_{k-1} \dots m_1 m_0})_2$ — двійкове представлення степеня n , тобто,

$$n = m_k \cdot 2^k + m_{k-1} \cdot 2^{k-1} + \dots + m_1 \cdot 2 + m_0,$$

де $m_k = 1, m_i \in \{0, 1\}$.

Тоді

$$x^n = x^{((\dots((m_k \cdot 2 + m_{k-1}) \cdot 2 + m_{k-2}) \cdot 2 + \dots) \cdot 2 + m_1) \cdot 2 + m_0} =$$

Таким чином, алгоритм повторюваного піднесення до квадрата зводиться до мультиплікативного аналогу схеми Горнера:

$$\left\{ \begin{array}{l} s_1 = x \\ s_{i+1} = s_i^2 \cdot x^{m_{k-i}} \\ i = 1, 2, \dots, k \end{array} \right\}.$$

3.3 Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити алгоритми теорії чисел: найбільший спільний дільник, найменше спільне кратне, решето Ератосфена, бінарне піднесення до ступеня.

2. Виконати індивідуальне завдання згідно варіанту (табл. 3.1-3.3).

3. Оформити звіт. Звіт з лабораторної роботи повинен містити: мету роботи; завдання на виконання роботи; текст програми; результати роботи; висновки.

4. Відповісти на контрольні запитання.

3.4 Індивідуальні завдання

Таблиця 3.1 – Завдання на тему «Найбільший спільний дільник та найменше спільне кратне, алгоритм Евкліда»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/uk/problems/1148
2	https://www.e-olymp.com/uk/problems/1151
3	https://www.e-olymp.com/uk/problems/1690
4	https://www.e-olymp.com/uk/problems/571
5	https://www.e-olymp.com/uk/problems/135
6	https://www.e-olymp.com/uk/problems/1154
7	https://www.e-olymp.com/uk/problems/1243
8	https://www.e-olymp.com/uk/problems/1181
9	https://www.e-olymp.com/uk/problems/1183
10	https://www.e-olymp.com/uk/problems/1155

Таблиця 3.2 – Завдання на тему «Прості числа, решето Ератосфена»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/uk/problems/33
2	https://www.e-olymp.com/uk/problems/4742
3	https://www.e-olymp.com/uk/problems/1302
4	https://www.e-olymp.com/uk/problems/1782
5	https://www.e-olymp.com/uk/problems/4076
6	https://www.e-olymp.com/uk/problems/2245
7	https://www.e-olymp.com/uk/problems/2612
8	https://www.e-olymp.com/uk/problems/3843
9	https://www.e-olymp.com/uk/problems/1904
10	https://www.e-olymp.com/uk/problems/22

Таблиця 3.3 – Завдання на тему «Модулярна арифметика, бінарне піднесення до ступеня»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/uk/problems/2814
2	https://www.e-olymp.com/uk/problems/480
3	https://www.e-olymp.com/uk/problems/273
4	https://www.e-olymp.com/uk/problems/965
5	https://www.e-olymp.com/uk/problems/5897

3.5 Контрольні запитання

1. Дайте визначення найбільшого спільного дільника.
2. Дайте визначення найменшого спільного кратного.
3. Наведіть алгоритм Евкліда.
4. Наведіть властивості НСК.
5. Як працює алгоритм "решето Ератосфена"?
6. Наведіть алгоритм бінарного піднесення до ступеня.

ЛАБОРАТОРНА РОБОТА №4 ДИНАМІЧНЕ ПРОГРАМУВАННЯ

4.1 Мета роботи

Вивчити основні принципи динамічного програмування.

4.2 Теоретичні відомості

У багатьох олімпіадних задачах з програмування розв'язок за допомогою рекурсії або повного перебору вимагає виконання дуже великої кількості операцій. Спроба вирішити такі завдання, наприклад, повним перебором, призводить до перевищення часу виконання.

Однак серед переборних і деяких інших завдань можна виділити клас задач, що володіють однією властивістю: маючи рішення деяких підзадач (наприклад, для меншого числа n), можна практично без перебору знайти рішення вихідної задачі.

Такі завдання вирішують методом динамічного програмування, а під самим динамічним програмуванням розуміють зведення задачі до підзадач.

Щоб краще зрозуміти суть динамічного програмування, спочатку слід більш формально визначити поняття задачі та підзадачі.

Нехай вихідна задача полягає в знаходженні деякого числа T при вихідних даних $n_1, n_2, \dots, n_k$. То, якщо можна говорити про функції $T(n_1, n_2, \dots, n_k)$, значення якої i є необхідна відповідь. Тоді підзадачами вважаються завдання $T(i_1, i_2, \dots, i_k)$ при $i_1 < n_1, i_2 < n_2, \dots, i_k < n_k$.

Далі будемо говорити про одновимірне та двовимірне динамічне програмування при $k = 1, k = 2$ відповідно.

Класичним завданням двовимірного динамічного програмування є задача про маршрути на прямокутному полі. У різних формулюваннях необхідно порахувати кількість маршрутів або знайти маршрут, який є кращим.

4.3 Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи динамічного програмування.
2. Виконати індивідуальне завдання згідно варіанту (табл. 4.1-4.2).
3. Оцінити обчислювальну складність алгоритму.
4. Оформити звіт з роботи.
5. Відповісти на контрольні запитання.

4.4 Індивідуальні завдання

Таблиця 4.1 – Завдання на тему «Одновимірне динамічне програмування»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/ru/problems/115
2	https://www.e-olymp.com/ru/problems/263
3	https://www.e-olymp.com/ru/problems/399
4	https://www.e-olymp.com/ru/problems/588
5	https://www.e-olymp.com/ru/problems/1281
6	https://www.e-olymp.com/ru/problems/5848
7	https://www.e-olymp.com/ru/problems/5849
8	https://www.e-olymp.com/ru/problems/4051
9	https://www.e-olymp.com/ru/problems/5846
10	https://www.e-olymp.com/ru/problems/2422

Таблиця 4.2 – Завдання на тему «Двовимірне динамічне програмування»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/ru/problems/15
2	https://www.e-olymp.com/ru/problems/5854
3	https://www.e-olymp.com/ru/problems/1704
4	https://www.e-olymp.com/ru/problems/4018
5	https://www.e-olymp.com/ru/problems/4019

4.5 Контрольні запитання

1. У чому полягає динамічне програмування?
2. Для розв'язання яких задач можна використовувати динамічне програмування?
3. З яких етапів складається процес розроблення алгоритмів динамічного програмування?
4. Які існують види динамічного програмування і чим вони відрізняються?
5. У чому полягає перекриття підзадач? Наведіть приклади.
6. Яким чином можна реалізувати запам'ятовування в процесі динамічного програмування?
7. Яким чином можна поліпшити час роботи розробленого Вами для розв'язання однієї із задач алгоритму динамічного програмування?

ЛАБОРАТОРНА РОБОТА №5 АЛГОРИТМИ ОБХОДУ ГРАФУ

5.1 Мета роботи

Вивчити алгоритми обходу графів на основі пошуку в ширину та на основі пошуку в глибину.

5.2 Теоретичні відомості

Існує багато алгоритмів на графах, які ґрунтуються на систематичному переборі їх вершин або обході вершин, під час якого кожна вершина отримує унікальний порядковий номер. Виділяється два основних алгоритми обходів графів або пошуку у графах – це пошук в глибину та пошук в ширину.

Пошук в глибину (Depth-first search, DFS) .

Нехай $G=(V, E)$ – простий зв'язний граф, усі вершини якого позначено попарно різними символами. У процесі пошуку в глибину вершинам графа G надають номери (DFS-номери) та певним чином позначають ребра. У ході роботи алгоритму використовують стек. DFS-номер вершини x позначають $DFS(x)$.

1. Почати з довільної вершини vs . Виконати $DFS(vs)=1$. Включити цю вершину в стек.

2. Розглянути вершину вгорі стеку: нехай це вершина x . Якщо всі ребра, інцидентні вершині x , позначено, то перейти до кроку 4, інакше – до кроку 3.

3. Нехай $\{x,y\}$ - непозначене ребро. Якщо $DFS(y)$ уже визначено, то визначити $DFS(y)$ як черговий DFS-номер, включити цю вершину в стек і перейти до кроку 2.

4. Виключити вершину x зі стеку. Якщо стек порожній, то зупинитись, інакше – перейти до кроку 2.

Щоб вибір номерів був однозначним, доцільно домовитись, що вершини, суміжні, з тією, яка вже отримала DFS-номер, аналізують за зростанням їх порядкових номерів (або в алфавітному порядку).

Пошук вшир (Breadth first search).

У процесі пошуку в ширину вершини графа проглядають в іншій послідовності, ніж у методі пошуку в глибину. BFS-номер вершини x позначають відповідно $BFS(x)$. Під час пошуку спочатку

проглядаються усі сусідні вершини, після цього – сусіди сусідів і так далі.

У ході реалізації алгоритму використовують структуру даних чергу. Що раніше відвідується вершина (включається в чергу), то раніше вона використовується (і виключається з черги). Використання вершини полягає в перегляді одразу всіх ще не відвіданих її сусідів.

Алгоритм пошуку в ширину у простому зв'язаному графі

1. Почати з довільної вершини v_s . Виконати $BFS(v)=1$. Включити вершину v у чергу.

2. Розглянути вершину, яка перебуває на початку черги; нехай це буде вершина x . Якщо для всіх вершин, суміжних із вершиною x , вже визначено BFS -номери, то перейти до кроку 4, інакше – до кроку 3.

3. Нехай $\{x, y\}$ – ребро, у якому номер $BFS(y)$ не визначено. - Визначити $BFS(y)$ як черговий BFS -номер, включити цю вершину у чергу й перейти до кроку 2.

4. Виключити вершину x із черги. Якщо черга порожня, то зупинитись, інакше – перейти до кроку 2.

Щоб результат виконання алгоритму був однозначним, вершини, які суміжні з вершиною x , аналізують за зростанням їх порядкових номерів (або в алфавітному порядку).

5.3 Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити алгоритми обходу графів на основі пошуку в ширину та на основі пошуку в глибину.

2. Виконати індивідуальне завдання згідно варіанту (табл. 5.1).

3. Оцінити обчислювальну складність алгоритму.

4. Оформити звіт з роботи.

5. Відповісти на контрольні запитання.

5.4 Індивідуальні завдання

Таблиця 5.1 – Завдання на тему «Алгоритми обходу графів на основі пошуку в ширину та на основі пошуку в глибину»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/uk/problems/4001
2	https://www.e-olymp.com/uk/problems/1061
3	https://www.e-olymp.com/uk/problems/122
4	https://www.e-olymp.com/uk/problems/1063
5	https://www.e-olymp.com/uk/problems/978
6	https://www.e-olymp.com/uk/problems/2401
7	https://www.e-olymp.com/uk/problems/982
8	https://www.e-olymp.com/uk/problems/993
9	https://www.e-olymp.com/uk/problems/3682
10	https://www.e-olymp.com/uk/problems/4050

5.5 Контрольні запитання

1. Алгоритм обходу графа на основі пошуку в ширину.
2. Алгоритм обходу графа на основі пошуку в глибину.

ЛАБОРАТОРНА РОБОТА №6

АЛГОРИТМИ ПОШУКУ НАЙКОРОТШИХ

ШЛЯХІВ У ГРАФАХ

6.1 Мета роботи

Вивчити основні алгоритми пошуку найкоротшого шляху в графах: алгоритми Дейкстри, Флойда-Воршелла та Беллмана-Форда

6.2 Теоретичні відомості

В теорії графів, задача про найкоротший шлях полягає в знаходженні такого шляху між двома вершинами (або вузлами) графу, що сума ваг ребер з яких він складається, мінімальна. Прикладом може бути знаходження найкоротшого шляху між двома пунктами на дорожній мапі; в цьому випадку, вершинами є пункти, а ребрами — відтинки дороги із вагами, рівними часу, необхідному для подолання цього відрізка.

Найважливіші алгоритми для розв'язання цієї задачі:

Алгоритм Дейкстри розв'язує задачі з однією парою, одним входом і одним виходом.

Алгоритм Флойда-Воршелла розв'язує задачу для всіх пар.

Алгоритм Беллмана-Форда розв'язує задачі з одним входом, якщо ваги ребер можуть бути від'ємні.

Алгоритм Дейкстри.

Формулювання задач про знаходження відстаней таке:

1. Для заданої початкової вершини а знайти найкоротші шляхи від а до всіх інших вершин.

2. Знайти найкоротші шляхи між усіма парами вершин.

Виявляється, що майже всі методи розв'язання задачі про найкоротший шлях від заданої початкової вершини до заданої кінцевої вершини, також дають змогу знайти й найкоротші шляхи від вершини а до всіх інших вершин графа. Отже, за їх допомогою можна розв'язати задачу 1 із невеликими додатковими обчислювальними витратами. З іншого боку, задачу 2 можна розв'язати або п разів застосувавши алгоритм задачі 1 із різними початковими вершинами, або один раз застосувавши спеціальний алгоритм.

Алгоритм Дейкстри – найефективніший алгоритм на графах.

Цей алгоритм знаходить найкоротшу відстань від однієї вершини графу до всіх решти. Алгоритм працює лише для графів у яких ребра не мають від'ємної ваги.

Алгоритм Форда-Бельмана

Так само як і алгоритм Дейкстри, алгоритм Форда-Беллмана обраховує у зваженому графі найкоротші шляхи від однієї вершини до всіх інших. Він підходить для роботи з графами, в яких є ребра від'ємної ваги. Але спектр використання алгоритму зачіпає не всі такі графи. Це пояснюється тим, що кожний наступний прохід по шляху, складеному з ребер, сума ваг яких є від'ємна, лише покращує задане значення. Безкінечне число покращень робить неможливим визначення одного конкретного значення, яке являлося б оптимальним. У зв'язку з цим алгоритм Форда-Беллмана не використовується з графами, у яких є від'ємні цикли, але він дозволяє визначити наявність таких.

Алгоритм Флойда-Воршелла — алгоритм динамічного програмування для знаходження найкоротших відстаней між усіма вершинами зваженого орієнтованого графа. Для знаходження найкоротших шляхів між всіма вершинами графа використовується не перебір всіх можливостей, що приведе до використання часу та затрат більшого об'єму пам'яті, а динамічне програмування, тобто підзадачі, які будуть потрібні для вирішення головної задачі, прораховуються завчасно і потім використовуються.

6.3 Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні алгоритми пошуку найкоротшого шляху в графах: алгоритми Дейкстри, Флойда-Воршелла та Беллмана-Форда.
2. Виконати індивідуальне завдання згідно варіанту (табл. 6.1).
3. Оцінити обчислювальну складність алгоритму.
4. Оформити звіт з роботи.
5. Відповісти на контрольні запитання.

6.4 Індивідуальні завдання

Таблиця 6.1 – Завдання на тему «Алгоритми пошуку найкоротшого шляху в графах»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/uk/problems/1453
2	https://www.e-olymp.com/uk/problems/974
3	https://www.e-olymp.com/uk/problems/975
4	https://www.e-olymp.com/uk/problems/1365
5	https://www.e-olymp.com/uk/problems/1388

6.5 Контрольні запитання

1. Для роботи з якими видами графів призначені алгоритми пошуку найкоротших шляхів?
2. За яких умов може використовуватися алгоритм Дейкстри?
3. У чому полягає алгоритм Дейкстри?
4. На основі яких структур даних побудований алгоритм Дейкстри?
5. Для розв'язання яких задач призначений алгоритм Дейкстри?
6. Для якого виду графів може застосовуватися алгоритм Флойда-Воршелла?
7. У чому полягає алгоритм Флойда-Воршелла?
8. У яких випадках можна застосовувати алгоритм Беллмана-Форда?
9. У чому полягає алгоритм Беллмана-Форда?

ЛАБОРАТОРНА РОБОТА №7

ДЕРЕВО ВІДРІЗКІВ

7.1 Мета роботи

Вивчити основні алгоритми використання дерев відрізків

7.2 Теоретичні відомості

Дерево відрізків – це структура даних, яка дозволяє ефективно (за асимптотику $O(\log n)$) реалізувати операції такого вигляду: знаходження суми / мінімуму елементів масиву в заданому відрізку ($a(l...r]$, де l і r надходять на вхід алгоритму), при цьому додатково можлива зміна елементів масиву: як зміна значення одного елемента, так і зміна елементів на цілому підвідрізку масиву (тобто дозволяється привласнити всім елементам $a(l...r]$ будь-яке значення, або додати до всіх елементів масиву якесь число).

Важливою особливістю дерев відрізків є те, що вони споживають лінійний обсяг пам'яті: стандартному дереву відрізків потрібно близько $4n$ елементів пам'яті для роботи над масивом розміру n .

7.3 Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи використання дерев відрізків.
2. Виконати індивідуальне завдання згідно варіанту (табл. 7.1).
3. Оцінити обчислювальну складність алгоритму.
4. Оформити звіт з роботи.
5. Відповісти на контрольні запитання.

7.4 Індивідуальні завдання

Таблиця 7.1 – Завдання на тему «Дерева відрізків»

№	Умова задачі (ресурс для перевірки проходження тестів)
1	https://www.e-olymp.com/uk/problems/695
2	https://www.e-olymp.com/uk/problems/2941
3	https://www.e-olymp.com/uk/problems/4481
4	https://www.e-olymp.com/uk/problems/4482
5	https://www.e-olymp.com/uk/problems/4483

7.5 Контрольні запитання

1. Поняття структури даних дерево відрізків.
2. Як виконується побудова дерева відрізків?
3. Яким чином виконується запит суми?
4. Яким чином виконується запит оновлення?
5. Яким чином виконується запит пошуку мінімуму або максимуму?

ЛІТЕРАТУРА

1. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms. – 3rd. – MIT Press, 2009.
2. Марієнко Л. Двійковий (binary) пошук в масивах в С [Електронний ресурс] / Л. Марієнко. – Режим доступу: <http://cppstudio.com/uk/post/2996/>.
3. Knuth, D. E. The Art of Computer Programming: Volume 1. Fundamental Algorithms. – 3rd. – Addison-Wesley Professional, 1995.
4. Knuth, D. E. The Art of Computer Programming: Volume 3. Sorting and Searching. – 2d. – Addison-Wesley Professional, 1998.
5. Теорія алгоритмів [Електронний ресурс]. – Режим доступу : <http://cybportal.univ.kiev.ua/wiki/>.
6. Sorting Algorithm Animations [Electronic resource]. – Access mode : <http://www.sorting-algorithms.com/>
7. Зиков О. О. Основи теорії графів. – М.: «Вузівська книга», 2004. – 664 с.
8. Базилевич Л. Є. Дискретна математика у прикладах і задачах : теорія множин, математична логіка, комбінаторика, теорія графів. – Математичний практикум. – Львів, 2013. – 486 с.